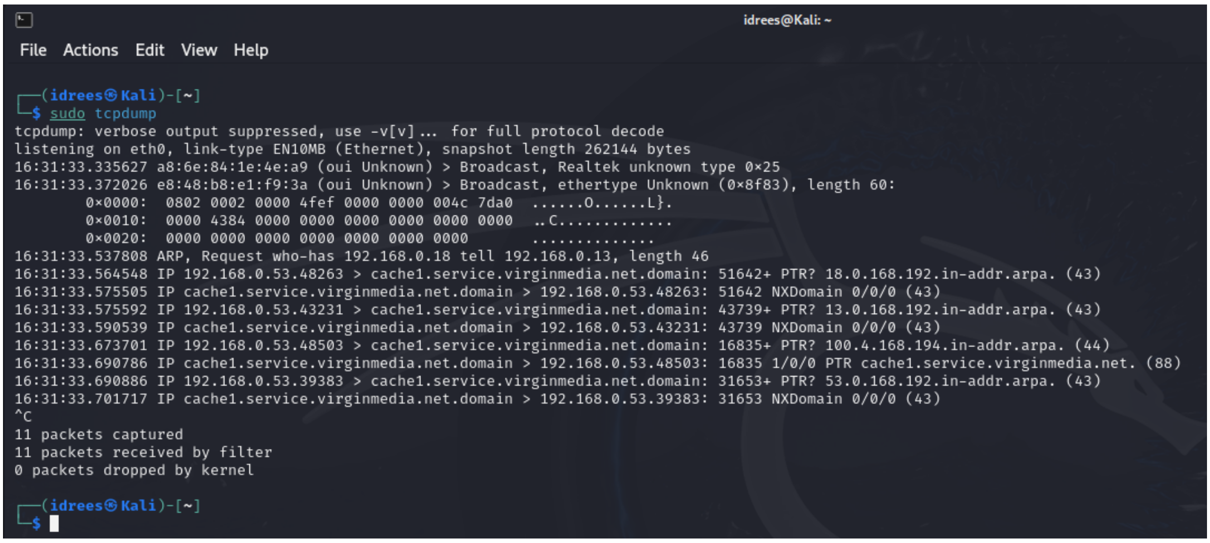
Contents

Scenario:

- Intermediate-level Network Administrator for an accounting firm
- Company suspects that some of its workstations are hacked.
- The job is to develop script that will run in the background and sample network traffic that meet certain criteria and save them for cyber forensic analysis.

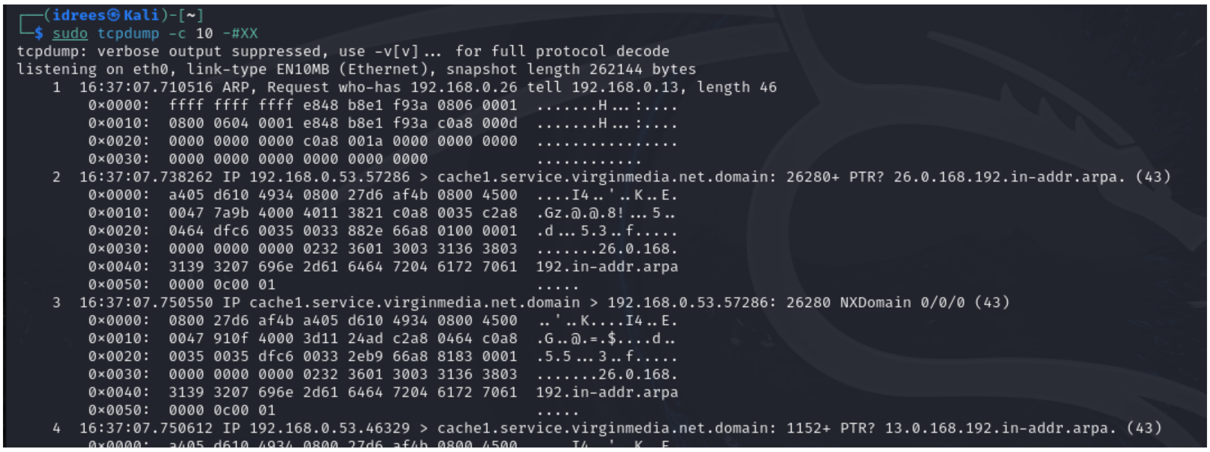
Task 1: Introduction and Getting Started

1. Run tcpdump on the Linux terminal.



```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:31:33.335627 a8:6e:84:1e:4e:a9 (oui Unknown) > Broadcast, Realtek unknown type 0x25
16:31:33.372026 e8:48:b8:e1:f9:3a (oui Unknown) > Broadcast, ethertype Unknown (0x8f83), length 60:
        0x0000:  0802 0002 0000 4fef 0000 0000 004c 7da0  ......O......L}.
        0x0010:  0000 4384 0000 0000 0000 0000 0000 0000  ..C.............
        0x0020:  0000 0000 0000 0000 0000 0000 0000       ..............
16:31:33.537808 ARP, Request who-has 192.168.0.18 tell 192.168.0.13, length 46
16:31:33.564548 IP 192.168.0.53.48263 > cache1.service.virginmedia.net.domain: 51642+ PTR? 18.0.168.192.in-addr.arpa. (43)
16:31:33.575505 IP cache1.service.virginmedia.net.domain > 192.168.0.53.48263: 51642 NXDomain 0/0/0 (43)
16:31:33.575592 IP 192.168.0.53.43231 > cache1.service.virginmedia.net.domain: 43739+ PTR? 13.0.168.192.in-addr.arpa. (43)
16:31:33.590539 IP cache1.service.virginmedia.net.domain > 192.168.0.53.43231: 43739 NXDomain 0/0/0 (43)
16:31:33.673701 IP 192.168.0.53.48503 > cache1.service.virginmedia.net.domain: 16835+ PTR? 100.4.168.194.in-addr.arpa. (44)
16:31:33.690786 IP cache1.service.virginmedia.net.domain > 192.168.0.53.48503: 16835 1/0/0 PTR cache1.service.virginmedia.net. (88)
16:31:33.690886 IP 192.168.0.53.39383 > cache1.service.virginmedia.net.domain: 31653+ PTR? 53.0.168.192.in-addr.arpa. (43)
16:31:33.701717 IP cache1.service.virginmedia.net.domain > 192.168.0.53.39383: 31653 NXDomain 0/0/0 (43)
^C
11 packets captured
11 packets received by filter
0 packets dropped by kernel

┌──(idrees㉿Kali)-[~]
└─$ █
```

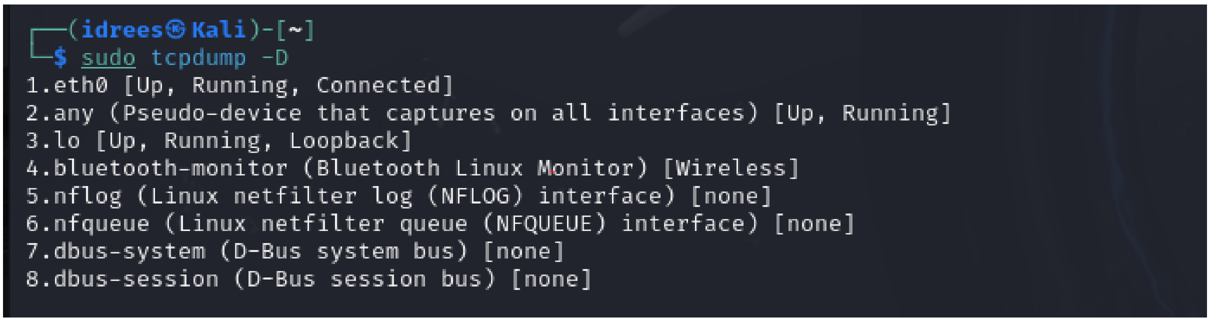
NOTE: tcpdump requires sudo permissions to be run. We can use the '-c' flag to limit the number of packets captured. Adding a '-#' flag will number the packets. Appending the 'A' will show the packets in ASCII and using 'XX' will show the data packets in hexadecimal and ASCII side by side. 'tttt' Will show the time, year month and day for each packet. 'D' flag will present all the interfaces that are installed in our system.



```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump -c 10 -#XX
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
    1  16:37:07.710516 ARP, Request who-has 192.168.0.26 tell 192.168.0.13, length 46
        0x0000:  ffff ffff ffff e848 b8e1 f93a 0806 0001  .......H...:....
        0x0010:  0800 0604 0001 e848 b8e1 f93a c0a8 000d  .......H...:....
        0x0020:  0000 0000 0000 c0a8 001a 0000 0000 0000  ................
        0x0030:  0000 0000 0000 0000 0000 0000            ............
    2  16:37:07.738262 IP 192.168.0.53.57286 > cache1.service.virginmedia.net.domain: 26280+ PTR? 26.0.168.192.in-addr.arpa. (43)
        0x0000:  a405 d610 4934 0800 27d6 af4b 0800 4500  ....I4..'..K..E.
        0x0010:  0047 7a9b 4000 4011 3821 c0a8 0035 c2a8  .Gz.@.@.8!....5..
        0x0020:  0464 dfc6 0035 0033 882e 66a8 0100 0001  .d....5.3..f.....
        0x0030:  0000 0000 0000 0232 3601 3003 3136 3803  .......26.0.168.
        0x0040:  3139 3207 696e 2d61 6464 7204 6172 7061  192.in-addr.arpa
        0x0050:  0000 0c00 01                             .....
    3  16:37:07.750550 IP cache1.service.virginmedia.net.domain > 192.168.0.53.57286: 26280 NXDomain 0/0/0 (43)
        0x0000:  0800 27d6 af4b a405 d610 4934 0800 4500  ..'..K....I4..E.
        0x0010:  0047 910f 4000 3d11 24ad c2a8 0464 c0a8  .G..@.=.$....d..
        0x0020:  0035 0035 dfc6 0033 2eb9 66a8 8183 0001  .5.5...3..f.....
        0x0030:  0000 0000 0000 0232 3601 3003 3136 3803  .......26.0.168.
        0x0040:  3139 3207 696e 2d61 6464 7204 6172 7061  192.in-addr.arpa
        0x0050:  0000 0c00 01                             .....
    4  16:37:07.750612 IP 192.168.0.53.46329 > cache1.service.virginmedia.net.domain: 1152+ PTR? 13.0.168.192.in-addr.arpa. (43)
        0x0000:  a405 d610 4934 0800 27d6 af4b 0800 4500  ....I4..'..K..E.
```

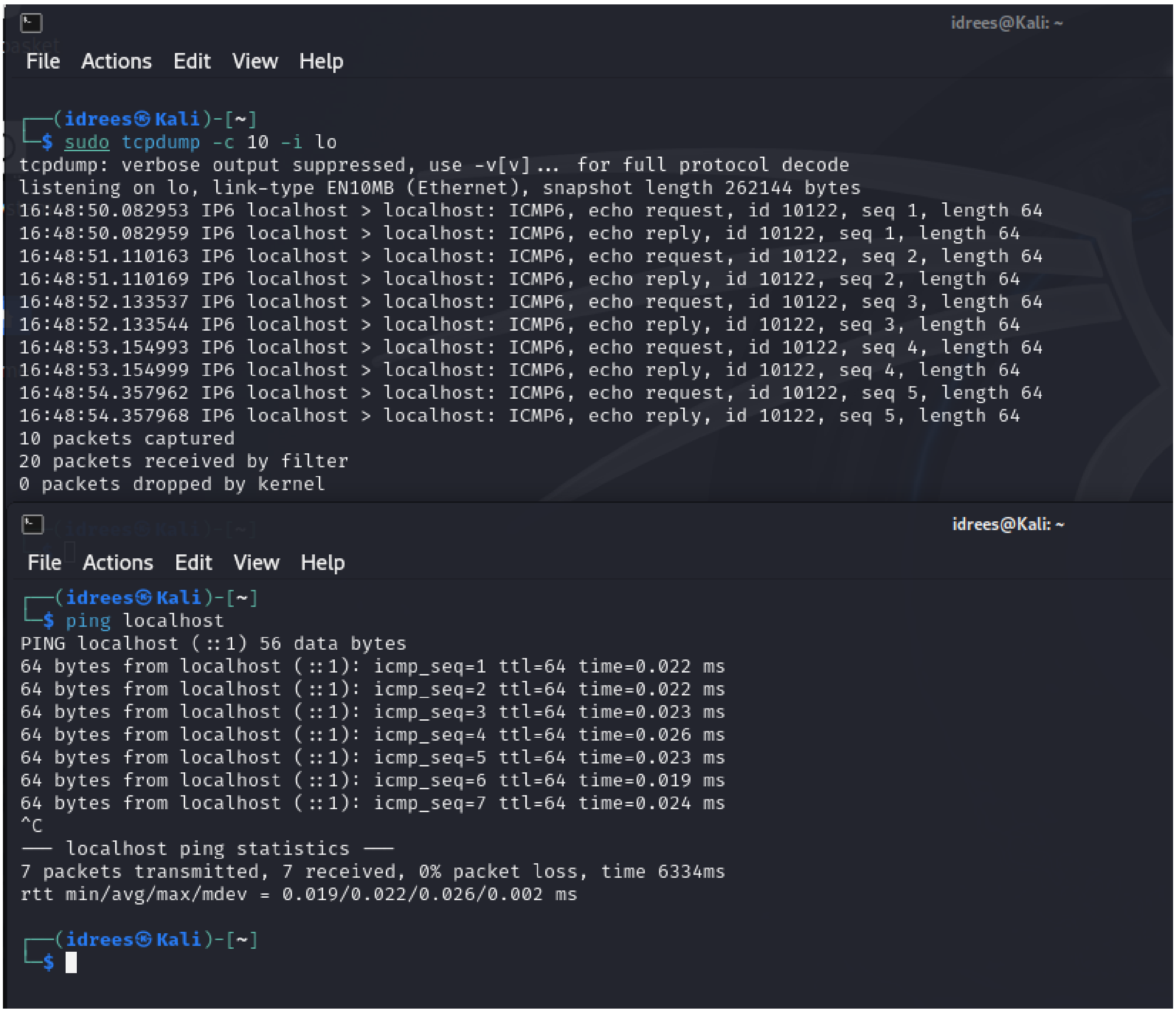
Task 2: Start Building the Logging Tool Script

1. Return a list of all the interfaces installed in the system.



```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump -D
1.eth0 [Up, Running, Connected]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
7.dbus-system (D-Bus system bus) [none]
8.dbus-session (D-Bus session bus) [none]
```

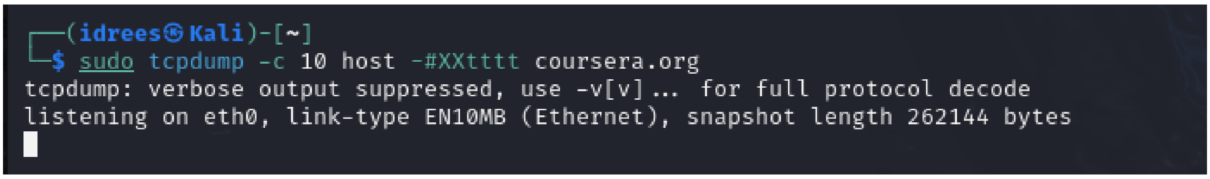
NOTE: We can specify what interface we want to capture packets from. Below I have pinged the local host and used tcpdump to capture all the pings. We can also specify the ports we want to capture from. We can also specify what website we want to capture traffic from using ‘src’, ‘dst’ or ‘host’ for both. Conditional filters can also be used (and, or).



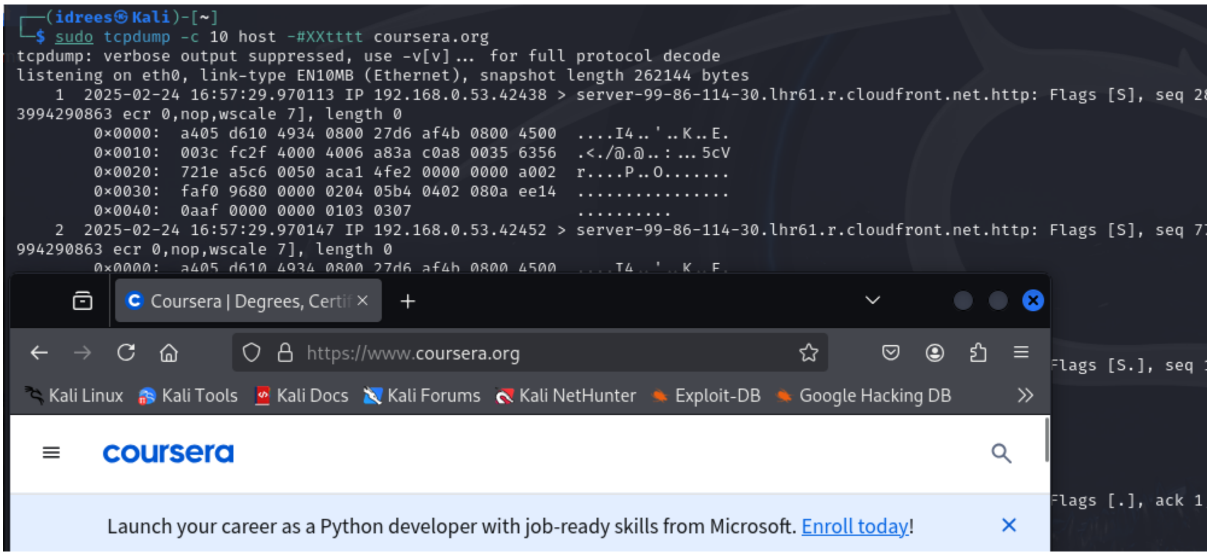
```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump -c 10 -i lo
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:48:50.082953 IP6 localhost > localhost: ICMP6, echo request, id 10122, seq 1, length 64
16:48:50.082959 IP6 localhost > localhost: ICMP6, echo reply, id 10122, seq 1, length 64
16:48:51.110163 IP6 localhost > localhost: ICMP6, echo request, id 10122, seq 2, length 64
16:48:51.110169 IP6 localhost > localhost: ICMP6, echo reply, id 10122, seq 2, length 64
16:48:52.133537 IP6 localhost > localhost: ICMP6, echo request, id 10122, seq 3, length 64
16:48:52.133544 IP6 localhost > localhost: ICMP6, echo reply, id 10122, seq 3, length 64
16:48:53.154993 IP6 localhost > localhost: ICMP6, echo request, id 10122, seq 4, length 64
16:48:53.154999 IP6 localhost > localhost: ICMP6, echo reply, id 10122, seq 4, length 64
16:48:54.357962 IP6 localhost > localhost: ICMP6, echo request, id 10122, seq 5, length 64
16:48:54.357968 IP6 localhost > localhost: ICMP6, echo reply, id 10122, seq 5, length 64
10 packets captured
20 packets received by filter
0 packets dropped by kernel
```

```
┌──(idrees㉿Kali)-[~]
└─$ ping localhost
PING localhost (::1) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.022 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.023 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.026 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.023 ms
64 bytes from localhost (::1): icmp_seq=6 ttl=64 time=0.019 ms
64 bytes from localhost (::1): icmp_seq=7 ttl=64 time=0.024 ms
^C
--- localhost ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6334ms
rtt min/avg/max/mdev = 0.019/0.022/0.026/0.002 ms

┌──(idrees㉿Kali)-[~]
└─$ 
```

2. Capture packets from coursera.org using tcpdump



```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump -c 10 host -#XXtttt coursera.org
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
█
```

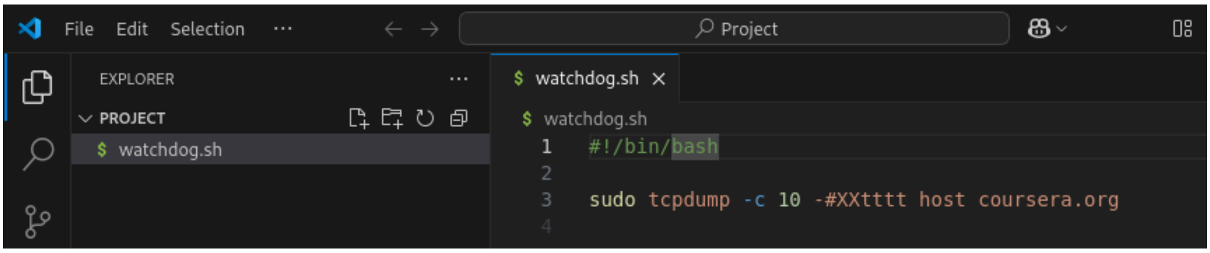


```
┌──(idrees㉿Kali)-[~]
└─$ sudo tcpdump -c 10 host -#XXtttt coursera.org
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
    1  2025-02-24 16:57:29.970113 IP 192.168.0.53.42438 > server-99-86-114-30.lhr61.r.cloudfront.net.http: Flags [S], seq 28
3994290863 ecr 0,nop,wscale 7], length 0
        0×0000:  a405 d610 4934 0800 27d6 af4b 0800 4500  ....I4..'..K..E.
        0×0010:  003c fc2f 4000 4006 a83a c0a8 0035 6356  .</@.@..:...5cV
        0×0020:  721e a5c6 0050 aca1 4fe2 0000 0000 a002  r....P..O.......
        0×0030:  faf0 9680 0000 0204 05b4 0402 080a ee14  ................
        0×0040:  0aaf 0000 0000 0103 0307                 ..........
    2  2025-02-24 16:57:29.970147 IP 192.168.0.53.42452 > server-99-86-114-30.lhr61.r.cloudfront.net.http: Flags [S], seq 7
994290863 ecr 0,nop,wscale 7], length 0
        0×0000:  a405 d610 4934 0800 27d6 af4b 0800 4500  ....I4..'..K..E.
Flags [S.], seq
Flags [.], ack 1
```

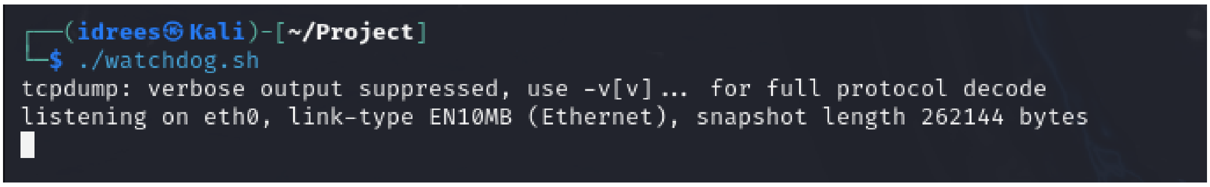
Launch your career as a Python developer with job-ready skills from Microsoft. Enroll today!

NOTE: 'host coursera.org' will return all packets that go to and from coursera.org. You can see that tcpdump will stay listening until coursera.org is searched.

3. Create the logging script and paste the command used in step 2 within VScode. When we run the script, it will execute that command



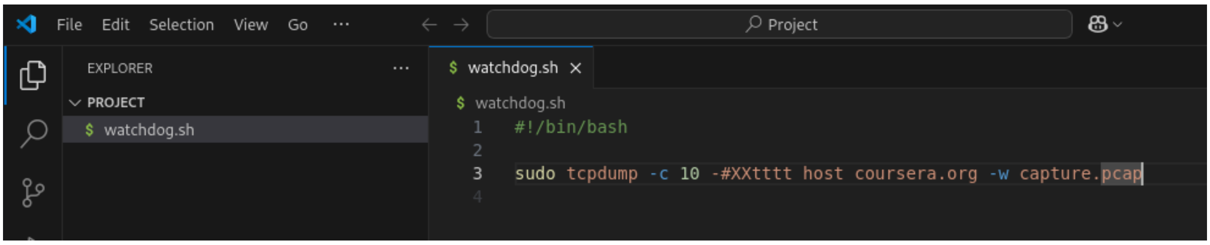
```
#!/bin/bash

sudo tcpdump -c 10 -#XXtttt host coursera.org
```



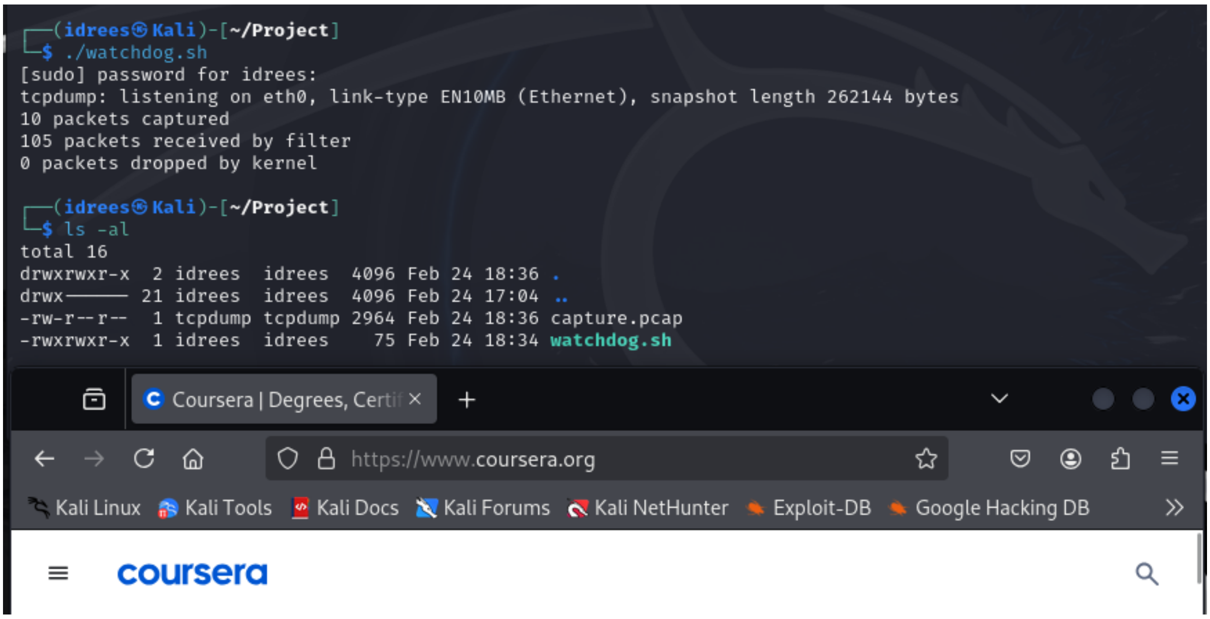
```
┌──(idrees㉿Kali)-[~/Project]
└─$ ./watchdog.sh
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
█
```

Task 3: Save Captured Packets in a Dump File

1. Record the results into a tcpdump capture file.

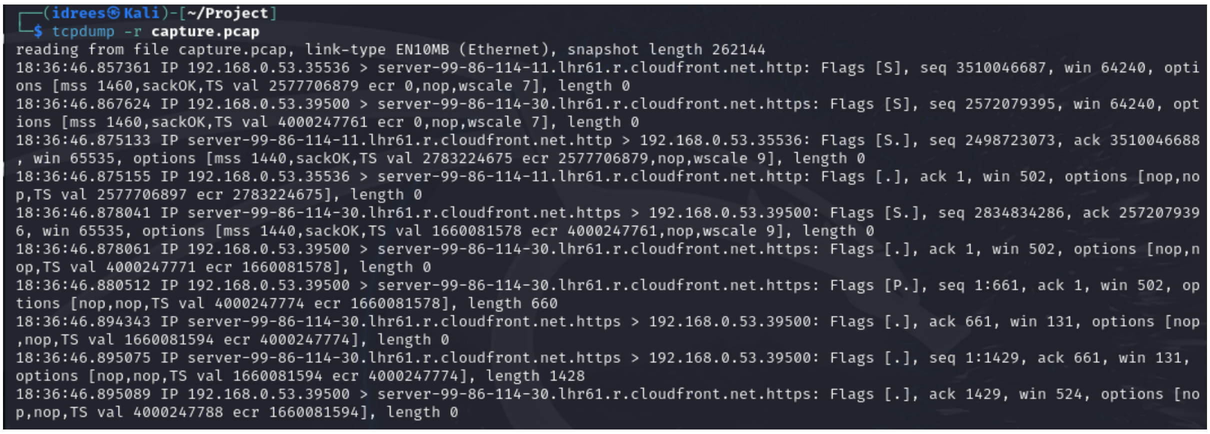


```
#!/bin/bash

sudo tcpdump -c 10 -#XXtttt host coursera.org -w capture.pcap
```



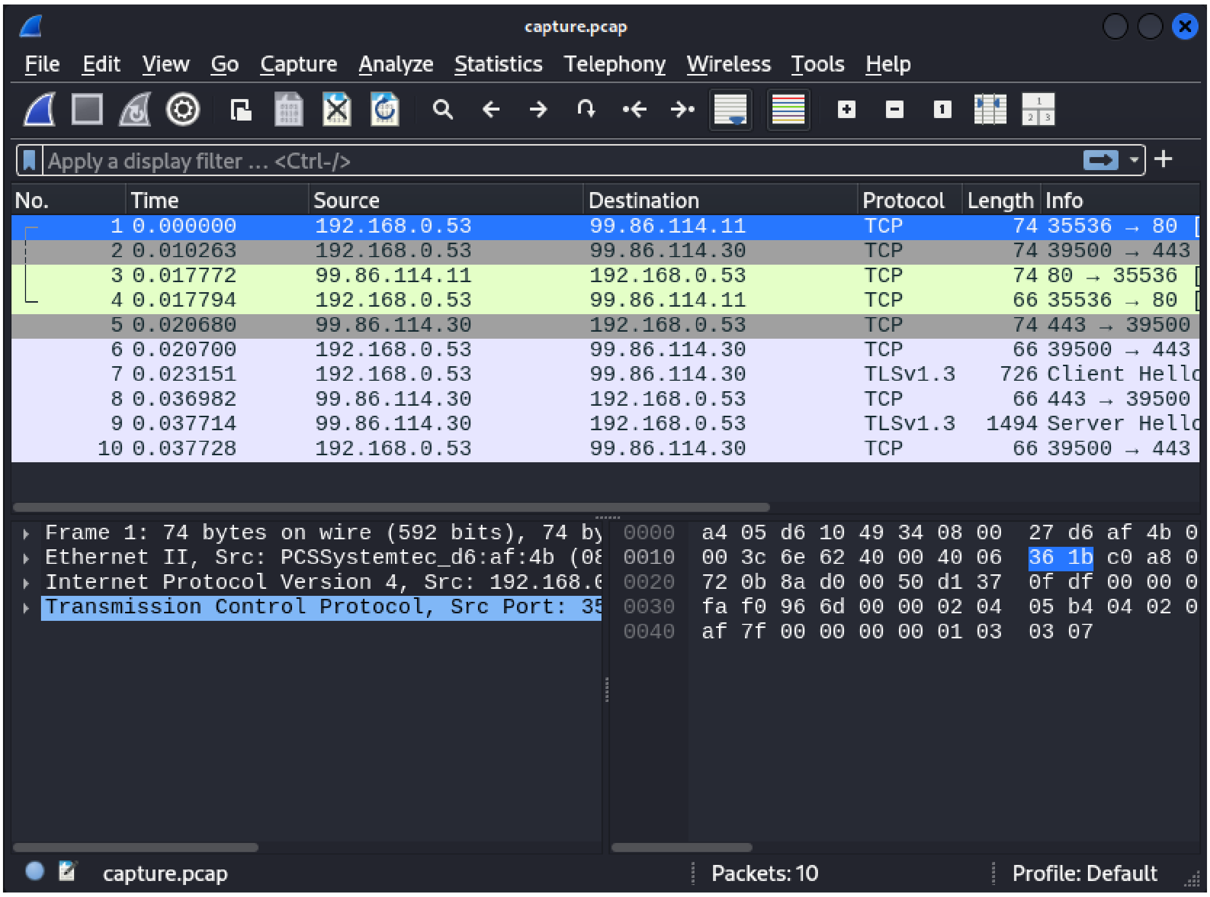
```
┌──(idrees㉿Kali)-[~/Project]
└─$ ./watchdog.sh
[sudo] password for idrees:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
10 packets captured
105 packets received by filter
0 packets dropped by kernel

┌──(idrees㉿Kali)-[~/Project]
└─$ ls -al
total 16
drwxrwxr-x  2 idrees  idrees  4096 Feb 24 18:36 .
drwx------ 21 idrees  idrees  4096 Feb 24 17:04 ..
-rw-r--r--  1 tcpdump tcpdump 2964 Feb 24 18:36 capture.pcap
-rwxrwxr-x  1 idrees  idrees    75 Feb 24 18:34 watchdog.sh
```

NOTE: When we run the file, we record all traffic from coursera.org and record it into the file capture.pcap. Important to allow write permissions for the file. The file stores the data in binary so we cannot read it. However, we can use tcpdump to read it. You can apply the same formatting flags mentioned before (-#).



```
┌──(idrees㉿Kali)-[~/Project]
└─$ tcpdump -r capture.pcap
reading from file capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
18:36:46.857361 IP 192.168.0.53.35536 > server-99-86-114-11.lhr61.r.cloudfront.net.http: Flags [S], seq 3510046687, win 64240, opti
ons [mss 1460,sackOK,TS val 2577706879 ecr 0,nop,wscale 7], length 0
18:36:46.867624 IP 192.168.0.53.39500 > server-99-86-114-30.lhr61.r.cloudfront.net.https: Flags [S], seq 2572079395, win 64240, opt
ions [mss 1460,sackOK,TS val 4000247761 ecr 0,nop,wscale 7], length 0
18:36:46.875133 IP server-99-86-114-11.lhr61.r.cloudfront.net.http > 192.168.0.53.35536: Flags [S.], seq 2498723073, ack 3510046688
, win 65535, options [mss 1440,sackOK,TS val 2783224675 ecr 2577706879,nop,wscale 9], length 0
18:36:46.875155 IP 192.168.0.53.35536 > server-99-86-114-11.lhr61.r.cloudfront.net.http: Flags [.], ack 1, win 502, options [nop,no
p,TS val 2577706897 ecr 2783224675], length 0
18:36:46.878041 IP server-99-86-114-30.lhr61.r.cloudfront.net.https > 192.168.0.53.39500: Flags [S.], seq 2834834286, ack 257207939
6, win 65535, options [mss 1440,sackOK,TS val 1660081578 ecr 4000247761,nop,wscale 9], length 0
18:36:46.878061 IP 192.168.0.53.39500 > server-99-86-114-30.lhr61.r.cloudfront.net.https: Flags [.], ack 1, win 502, options [nop,n
op,TS val 4000247771 ecr 1660081578], length 0
18:36:46.880512 IP 192.168.0.53.39500 > server-99-86-114-30.lhr61.r.cloudfront.net.https: Flags [P.], seq 1:661, ack 1, win 502, op
tions [nop,nop,TS val 4000247774 ecr 1660081578], length 660
18:36:46.894343 IP server-99-86-114-30.lhr61.r.cloudfront.net.https > 192.168.0.53.39500: Flags [.], ack 661, win 131, options [nop
,nop,TS val 1660081594 ecr 4000247774], length 0
18:36:46.895075 IP server-99-86-114-30.lhr61.r.cloudfront.net.https > 192.168.0.53.39500: Flags [.], seq 1:1429, ack 661, win 131,
options [nop,nop,TS val 1660081594 ecr 4000247774], length 1428
18:36:46.895089 IP 192.168.0.53.39500 > server-99-86-114-30.lhr61.r.cloudfront.net.https: Flags [.], ack 1429, win 524, options [no
p,nop,TS val 4000247788 ecr 1660081594], length 0
```

2. Use Wireshark to also view the tcpdump capture file.


capture.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
capture.pcap
Packets: 10
Profile: Default

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.53	99.86.114.11	TCP	74	35536 → 80 [
2	0.010263	192.168.0.53	99.86.114.30	TCP	74	39500 → 443
3	0.017772	99.86.114.11	192.168.0.53	TCP	74	80 → 35536 [
4	0.017794	192.168.0.53	99.86.114.11	TCP	66	35536 → 80 [
5	0.020680	99.86.114.30	192.168.0.53	TCP	74	443 → 39500
6	0.020700	192.168.0.53	99.86.114.30	TCP	66	39500 → 443
7	0.023151	192.168.0.53	99.86.114.30	TLSv1.3	726	Client Hello
8	0.036982	99.86.114.30	192.168.0.53	TCP	66	443 → 39500
9	0.037714	99.86.114.30	192.168.0.53	TLSv1.3	1494	Server Hello
10	0.037728	192.168.0.53	99.86.114.30	TCP	66	39500 → 443

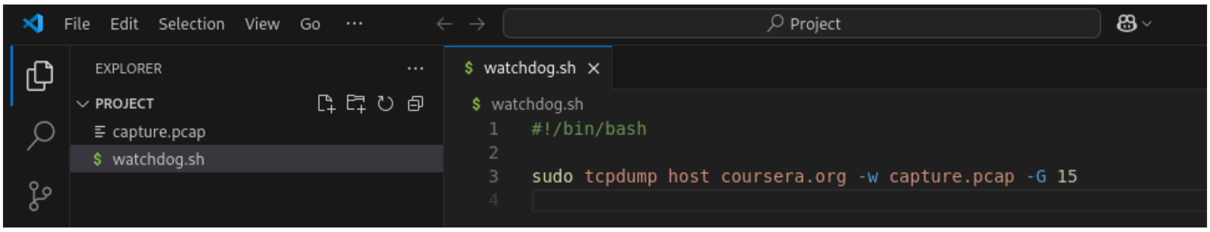
```
› Frame 1: 74 bytes on wire (592 bits), 74 by
› Ethernet II, Src: PCSSystemtec_d6:af:4b (08
› Internet Protocol Version 4, Src: 192.168.0
› Transmission Control Protocol, Src Port: 35
```

```
0000  a4 05 d6 10 49 34 08 00  27 d6 af 4b 0
0010  00 3c 6e 62 40 00 40 06  36 1b c0 a8 0
0020  72 0b 8a d0 00 50 d1 37  0f df 00 00 0
0030  fa f0 96 6d 00 00 02 04  05 b4 04 02 0
0040  af 7f 00 00 00 00 01 03  03 07
```

NOTE: On the bottom of the Wireshark window, it will format the packet in the TCP protocol. It will present it in a form that you can read.

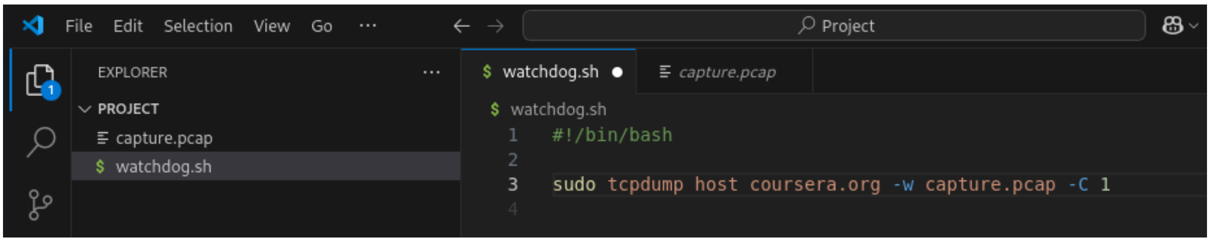
Task 4: Create Sequenced Dump Files

1. Write a new script that will wipe out the capture file every 15 seconds and write a new one.

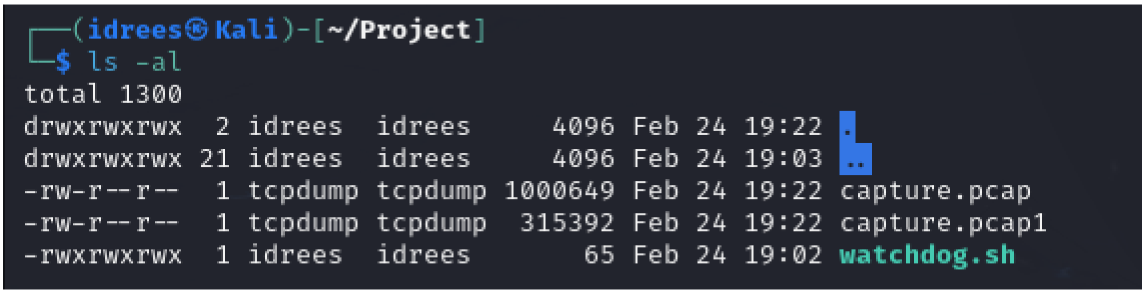


```
#!/bin/bash

sudo tcpdump host coursera.org -w capture.pcap -G 15
```

2. Set a size limit of 1mb, once the limit is met it will create a new file.



```
#!/bin/bash

sudo tcpdump host coursera.org -w capture.pcap -C 1
```

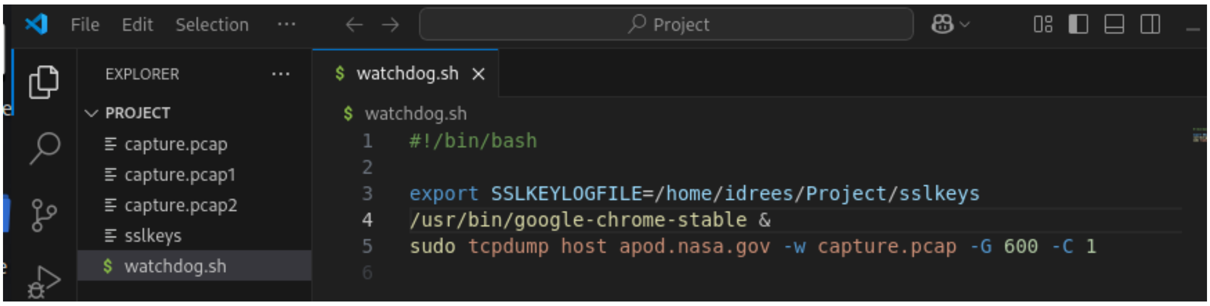


```
┌──(idrees㉿Kali)-[~/Project]
└─$ ls -al
total 1300
drwxrwxrwx  2 idrees  idrees     4096 Feb 24 19:22 .
drwxrwxrwx 21 idrees  idrees     4096 Feb 24 19:03 ..
-rw-r--r--  1 tcpdump tcpdump 1000649 Feb 24 19:22 capture.pcap
-rw-r--r--  1 tcpdump tcpdump  315392 Feb 24 19:22 capture.pcap1
-rwxrwxrwx  1 idrees  idrees       65 Feb 24 19:02 watchdog.sh
```

NOTE: We can see when capture.pcap exceeds a file size of 1000000, all data packets captured are stored in capture.pcap1. You can now combine both conditions the time limit and the file size together. This means you do not have to deal with one big file.

Task 5: Decrypt and Analyse Captured Traffic

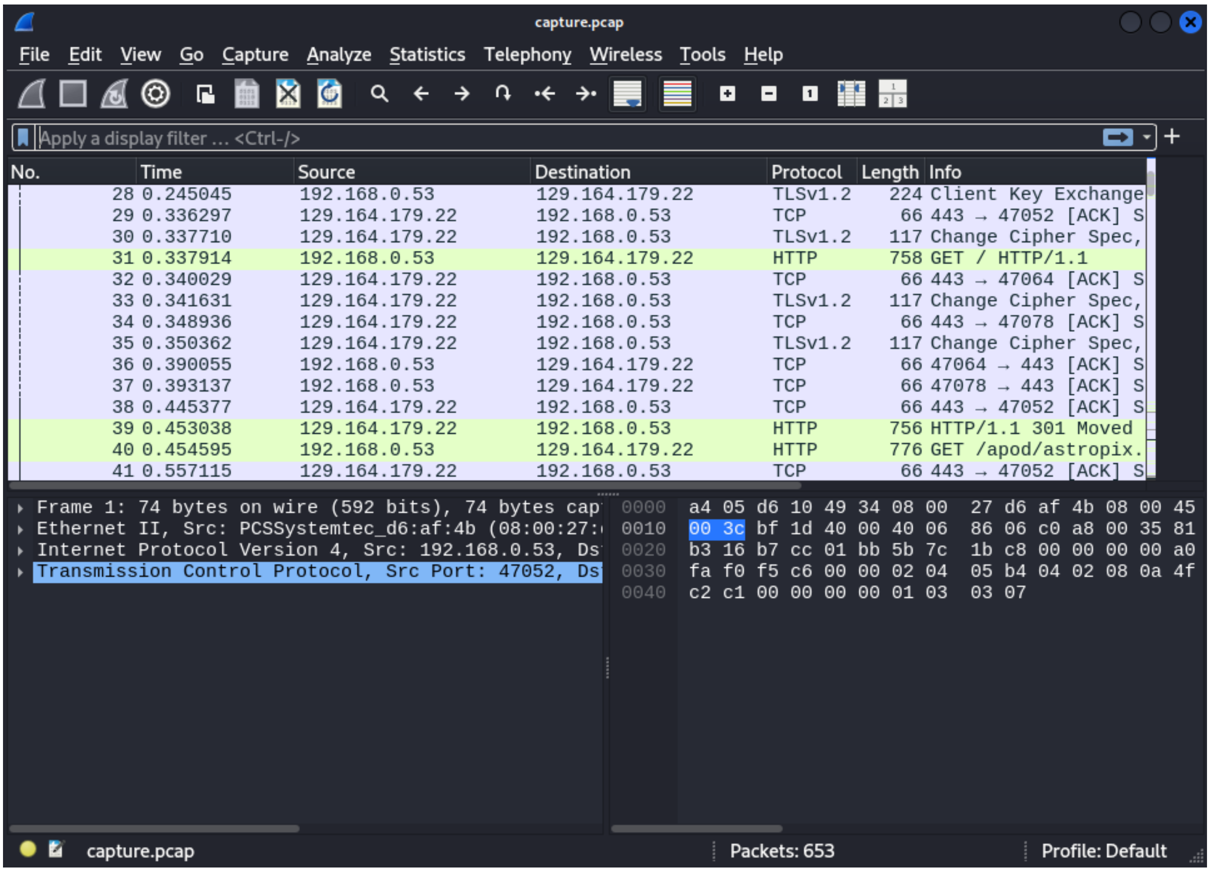
1. Write a new script to capture the public and the private key from the browser. TCP dump will capture the encrypted data



```
#!/bin/bash

export SSLKEYLOGFILE=/home/idrees/Project/sslkeys
/usr/bin/google-chrome-stable &
sudo tcpdump host apod.nasa.gov -w capture.pcap -G 600 -C 1
```

NOTE: This script sets an environment variable SSLKEYLOGFILE that Google Chrome will use log TLS/SSL sessions keys. We will run google chrome, so that it will write SSL/TLS keys to the specified file as it establishes secure connections and run tcpdump to capture traffic to/from the apos.nasa.gov website.

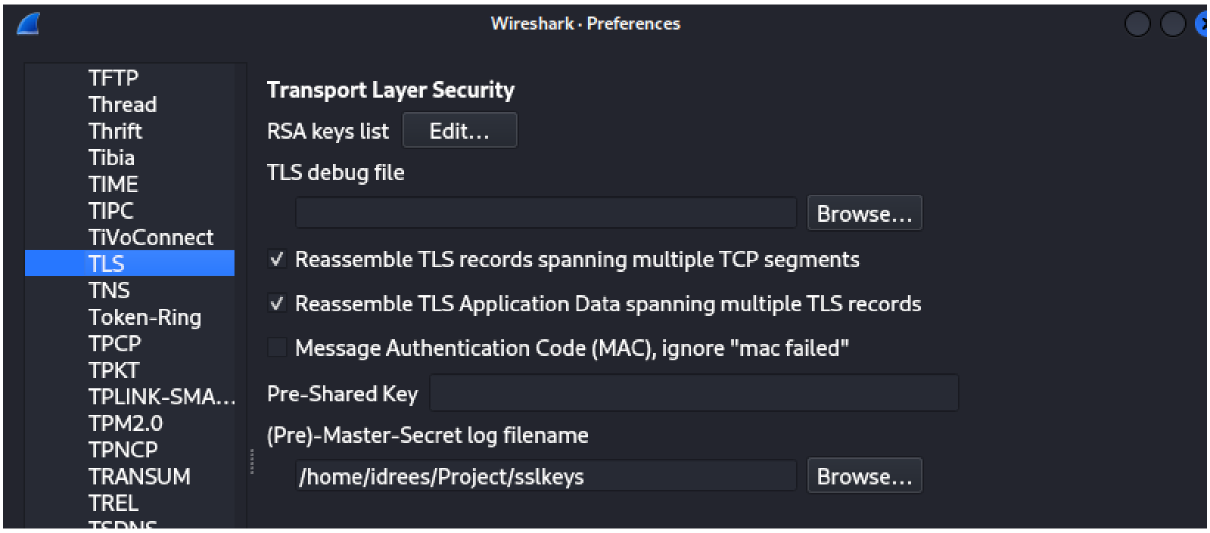
2. Use Wireshark to recreate the original unencrypted data



```
No.      Time       Source          Destination     Protocol  Length Info
      28 0.245045   192.168.0.53    129.164.179.22  TLSv1.2     224 Client Key Exchange
      29 0.336297   129.164.179.22  192.168.0.53    TCP          66 443 → 47052 [ACK] S
      30 0.337710   129.164.179.22  192.168.0.53    TLSv1.2     117 Change Cipher Spec,
      31 0.337914   192.168.0.53    129.164.179.22  HTTP        758 GET / HTTP/1.1
      32 0.340029   129.164.179.22  192.168.0.53    TCP          66 443 → 47064 [ACK] S
      33 0.341631   129.164.179.22  192.168.0.53    TLSv1.2     117 Change Cipher Spec,
      34 0.348936   129.164.179.22  192.168.0.53    TCP          66 443 → 47078 [ACK] S
      35 0.350362   129.164.179.22  192.168.0.53    TLSv1.2     117 Change Cipher Spec,
      36 0.390055   192.168.0.53    129.164.179.22  TCP          66 47064 → 443 [ACK] S
      37 0.393137   192.168.0.53    129.164.179.22  TCP          66 47078 → 443 [ACK] S
      38 0.445377   129.164.179.22  192.168.0.53    TCP          66 443 → 47052 [ACK] S
      39 0.453038   129.164.179.22  192.168.0.53    HTTP        756 HTTP/1.1 301 Moved
      40 0.454595   192.168.0.53    129.164.179.22  HTTP        776 GET /apod/astropix.
      41 0.557115   129.164.179.22  192.168.0.53    TCP          66 443 → 47052 [ACK] S
```

```
› Frame 1: 74 bytes on wire (592 bits), 74 bytes cap
› Ethernet II, Src: PCSSystemtec_d6:af:4b (08:00:27:
› Internet Protocol Version 4, Src: 192.168.0.53, Ds
› Transmission Control Protocol, Src Port: 47052, Ds
```

```
0000  a4 05 d6 10 49 34 08 00  27 d6 af 4b 08 00 45
0010  00 3c bf 1d 40 00 40 06  86 06 c0 a8 00 35 81
0020  b3 16 b7 cc 01 bb 5b 7c  1b c8 00 00 00 00 a0
0030  fa f0 f5 c6 00 00 02 04  05 b4 04 02 08 0a 4f
0040  c2 c1 00 00 00 00 01 03  03 07
```



Transport Layer Security

RSA keys list

TLS debug file

☑ Reassemble TLS records spanning multiple TCP segments

☑ Reassemble TLS Application Data spanning multiple TLS records

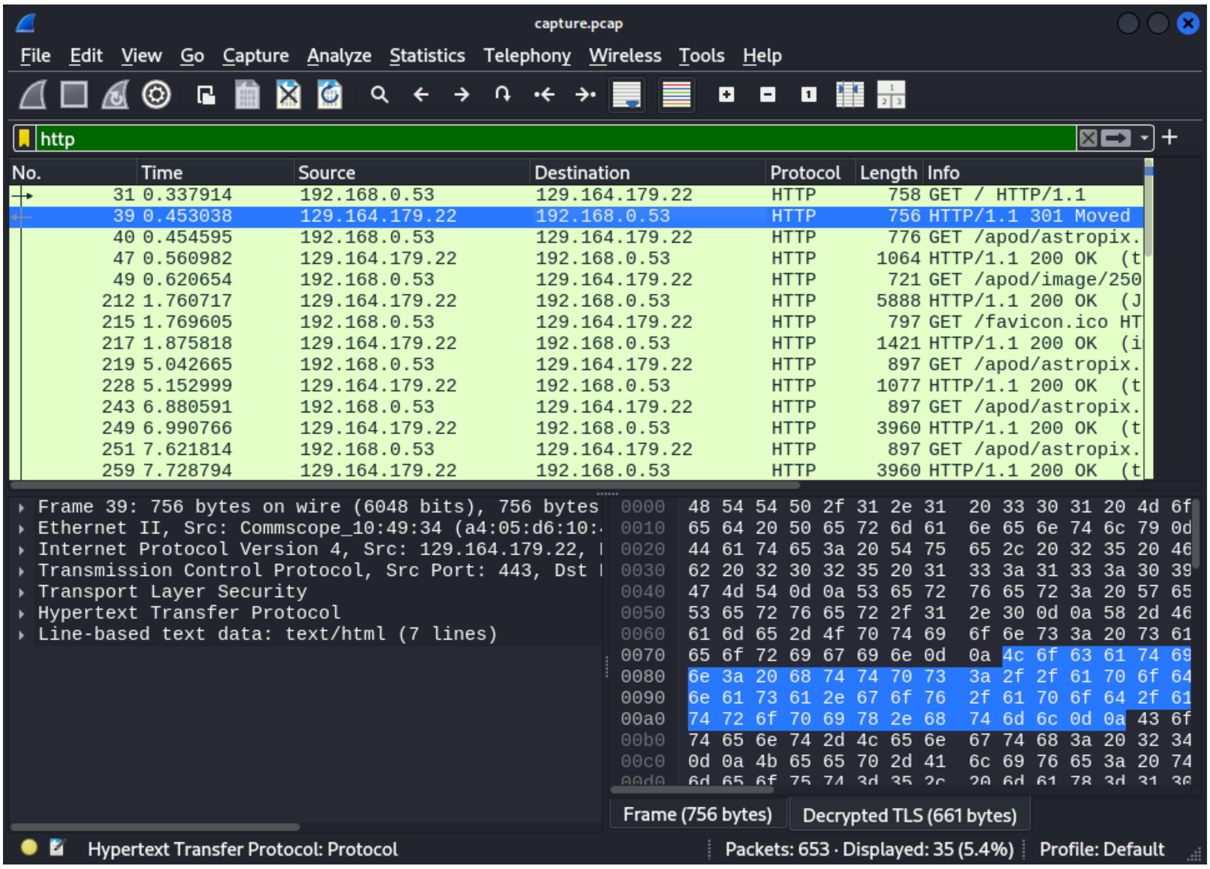
□ Message Authentication Code (MAC), ignore "mac failed"

Pre-Shared Key

(Pre)-Master-Secret log filename: /home/idrees/Project/sslkeys

NOTE: Opening the capture.pcap file and entering the sslkeys file into the Master-Secret log filename, we can view decrypted data in Wireshark (highlighted green).

3. Filter all http packets


capture.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

```
No.   Time       Source           Destination      Protocol Length Info
   31 0.337914   192.168.0.53     129.164.179.22   HTTP      758 GET / HTTP/1.1
   39 0.453038   129.164.179.22   192.168.0.53     HTTP      756 HTTP/1.1 301 Moved
   40 0.454595   192.168.0.53     129.164.179.22   HTTP      776 GET /apod/astropix.
   47 0.560982   129.164.179.22   192.168.0.53     HTTP     1064 HTTP/1.1 200 OK  (t
   49 0.620654   192.168.0.53     129.164.179.22   HTTP      721 GET /apod/image/250
  212 1.760717   129.164.179.22   192.168.0.53     HTTP     5888 HTTP/1.1 200 OK  (J
  215 1.769605   192.168.0.53     129.164.179.22   HTTP      797 GET /favicon.ico HT
  217 1.875818   129.164.179.22   192.168.0.53     HTTP     1421 HTTP/1.1 200 OK  (i
  219 5.042665   192.168.0.53     129.164.179.22   HTTP      897 GET /apod/astropix.
  228 5.152999   129.164.179.22   192.168.0.53     HTTP     1077 HTTP/1.1 200 OK  (t
  243 6.880591   192.168.0.53     129.164.179.22   HTTP      897 GET /apod/astropix.
  249 6.990766   129.164.179.22   192.168.0.53     HTTP     3960 HTTP/1.1 200 OK  (t
  251 7.621814   192.168.0.53     129.164.179.22   HTTP      897 GET /apod/astropix.
  259 7.728794   129.164.179.22   192.168.0.53     HTTP     3960 HTTP/1.1 200 OK  (t
```

```
› Frame 39: 756 bytes on wire (6048 bits), 756 bytes
› Ethernet II, Src: Commscope_10:49:34 (a4:05:d6:10:
› Internet Protocol Version 4, Src: 129.164.179.22,
› Transmission Control Protocol, Src Port: 443, Dst
› Transport Layer Security
› Hypertext Transfer Protocol
› Line-based text data: text/html (7 lines)
```

```
0000  48 54 54 50 2f 31 2e 31  20 33 30 31 20 4d 6f
0010  65 64 20 50 65 72 6d 61  6e 65 6e 74 6c 79 0d
0020  44 61 74 65 3a 20 54 75  65 2c 20 32 35 20 46
0030  62 20 32 30 32 35 20 31  33 3a 31 33 3a 30 39
0040  47 4d 54 0d 0a 53 65 72  76 65 72 3a 20 57 65
0050  53 65 72 76 65 72 2f 31  2e 30 0d 0a 58 2d 46
0060  61 6d 65 2d 4f 70 74 69  6f 6e 73 3a 20 73 61
0070  65 6f 72 69 67 69 6e 0d  0a 4c 6f 63 61 74 69
0080  6e 3a 20 68 74 74 70 73  3a 2f 2f 61 70 6f 64
0090  6e 61 73 61 2e 67 6f 76  2f 61 70 6f 64 2f 61
00a0  74 72 6f 70 69 78 2e 68  74 6d 6c 0d 0a 43 6f
00b0  74 65 6e 74 2d 4c 65 6e  67 74 68 3a 20 32 34
00c0  0d 0a 4b 65 65 70 2d 41  6c 69 76 65 3a 20 74
00d0  6d 65 6f 75 74 3d 35 2c  20 6d 61 78 3d 31 30
```

Frame (756 bytes) Decrypted TLS (661 bytes)

Hypertext Transfer Protocol: Protocol Packets: 653 · Displayed: 35 (5.4%) Profile: Default

NOTE: We can now see all the decrypted data. Wireshark does this for us and we can see it on the bottom right.